ChatGPT

Kiến trúc tổng thể của hệ thống

Ứng dụng bao gồm các thành phần chính: *Frontend* (giao diện ReactJS để xây dựng workflow), *Backend* (Node.js cung cấp API xử lý tin nhắn) và *Database* (PostgreSQL lưu trữ workflow và trạng thái hội thoại). Ngoài ra, **n8n** đóng vai trò trung gian xử lý sự kiện Messenger – nó nhận tin nhắn từ người dùng, gọi API của backend và gửi lại tin nhắn phản hồi. Dòng chảy dữ liệu có thể mô tả như sau: Người dùng gửi tin nhắn → n8n nhận tin nhắn (trigger) → n8n gọi API `/api/message` của Backend → Backend truy vấn/đọc workflow và trạng thái hiện tại → Backend trả về tin nhắn tiếp theo → n8n gửi tin nhắn đó cho người dùng. Kiến trúc tổng thể có thể tóm tắt:

- **Giao diện người dùng (ReactJS)**: Xây dựng công cụ **kéo-thả (drag-and-drop)** để tạo và chỉnh sửa luồng hội thoại. Ví dụ, thư viện React Flow chuyên hỗ trợ **UI dạng node-based editor** rất phù hợp cho việc này [1]. React Flow cung cấp một đối tượng JSON đại diện cho luồng (mảng `nodes` và `edges`) để có thể lưu trữ dễ dàng vào cơ sở dữ liệu [2].
- **Backend (Node.js, Express/NestJS)**: Cung cấp API RESTful để lưu/lấy dữ liệu workflow, và xử lý tin nhắn từ n8n. Ví dụ, có thể dùng **NestJS** với Typescript để tổ chức code theo mô hình module-controller-service (theo "clean architecture" [3]) nhằm tách biệt rõ logic nghiệp vụ. Backend sẽ duy trì *state* của mỗi cuộc hội thoại (mỗi user) trong DB hoặc cache. Backend cũng sẽ lưu workflow JSON do frontend tạo ra (xem phần dưới).
- **Cơ sở dữ liệu (PostgreSQL)**: Lưu trữ thông tin workflow và trạng thái hội thoại. Workflow được lưu dưới dạng **JSON (JSONB)** để dễ lưu trữ cấu trúc cây hoặc đồ thị. PostgreSQL hỗ trợ kiểu `jsonb` rất mạnh (hỗ trợ chỉ mục GIN, truy vấn nhanh) và được khuyến nghị dùng cho dữ liệu JSON [4]. Có thể dùng một bảng như `workflows(id, user_id, name, data JSONB)` để lưu luồng cho mỗi chatbot do người dùng tạo, và một bảng `conversations(user_id, flow_id, current_node_id, data JSONB, updated_at)` để lưu trạng thái hội thoại hiện tại.
- **n8n (Automation)**: Đã cấu hình sẵn để kết nối Facebook Messenger. Khi n8n nhận tin nhắn (thông qua Facebook Messenger node của n8n), nó sẽ gọi **Webhook/HTTP Request** tới API của chúng ta (ví dụ `POST /api/message`). Sau khi nhận response từ backend (dạng JSON chứa tin nhắn và lựa chọn tiếp theo), n8n sẽ dùng node trả tin (Facebook Messenger node) gửi tin nhắn đó đến người dùng.

[1] [2]

Lưu trữ workflow JSON dạng cây trong PostgreSQL

Workflow do người dùng thiết kế (câu hỏi/đáp án và các luồng nhánh) sẽ được serialise thành JSON. Một cấu trúc phổ biến là dùng các **node** và **edge** giống React Flow: mỗi node có `id`, `data` (chứa câu hỏi hoặc đáp án) và các đường nối (`edges`) định nghĩa luồng tiếp theo. Ví dụ React Flow định nghĩa một object `ReactFlowJsonObject` gồm `{ nodes: Node[], edges: Edge[] }` để lưu trữ dễ dàng vào database [2]. Ta có thể tạo một bảng trong PostgreSQL như:

```
CREATE TABLE workflows (
  id SERIAL PRIMARY KEY,
  owner_id UUID REFERENCES users(id),
  name TEXT,
```

```
  data JSONB,       -- lưu trữ toàn bộ cấu trúc workflow
  created_at TIMESTAMP DEFAULT NOW()
);
```

Ở đây `data JSONB` chứa đối tượng JSON (ví dụ giống React Flow JSON) đại diện cho cây/lưới luồng hội thoại. PostgreSQL lưu `jsonb` ở dạng nhị phân (binary JSON) có hiệu năng cao, hỗ trợ chỉ mục GIN và truy vấn linh hoạt [4]. Ta nên tạo *index GIN* trên cột `data` nếu cần query theo khóa nội dung, tuy thường với chatbot ta chỉ cần truy xuất toàn bộ đối tượng JSON. Khi cập nhật workflow, ta có thể ghi đè toàn bộ JSON hoặc dùng hàm jsonb_set của Postgres.

Trong JSON lưu trữ, mỗi node có thể chứa thông tin:
- `id`: định danh duy nhất
- `type`: loại node (ví dụ "question", "answer", "api call" v.v.)
- `text` / `content`: nội dung tin nhắn sẽ gửi (câu hỏi hoặc hướng dẫn).
- `options` / `edges`: mảng các lựa chọn dẫn đến node con (mỗi phần tử chứa `label` + `nextNodeId`).

Sử dụng JSON như vậy giúp linh hoạt mở rộng luồng mà không cần thay đổi cấu trúc bảng. Và vì là JSON "tree" (cây/đồ thị hướng), ta có thể biểu diễn luồng đa nhánh. Ví dụ:

```
{
  "nodes": [
    { "id": "1", "type": "bot", "text": "Xin chào! Bạn cần hỗ trợ gì?",
"edges": [ { "label": "Tư vấn", "next": "2" }, { "label": "Mua hàng",
"next": "3" } ] },
    { "id": "2", "type": "bot", "text": "Bạn muốn được tư vấn về sản phẩm
nào?", "edges": [ ... ] },
    { "id": "3", "type": "bot", "text": "Bạn đang quan tâm mặt hàng nào?",
"edges": [ ... ] }
  ]
}
```

Bằng cách lưu trữ dưới `JSONB`, việc mở rộng các nhánh hoặc thêm trường mới đều chỉ cần chỉnh JSON mà không cần migrate cơ sở dữ liệu. Điều quan trọng là khi truy vấn, backend sẽ lấy toàn bộ JSON (workflow) từ DB theo `flow_id` tương ứng, và duyệt/truy xuất trong đó để xác định câu trả lời tiếp theo.

[4]

Xử lý logic tin nhắn theo workflow và theo dõi trạng thái hội thoại

Quy trình xử lý tin nhắn: Khi backend nhận một HTTP request từ n8n (gồm `userId` và `text` người dùng gửi), nó sẽ thực hiện:
1. **Lấy trạng thái hiện tại**: Dựa vào `userId` (có thể kèm thông tin `flowId` nếu người dùng có nhiều chatbot), backend tìm bản ghi `conversation` tương ứng để biết user đang ở node nào trong workflow. Nếu chưa có (lần đầu), khởi tạo node hiện tại là nút gốc (root) của workflow.
2. **Xác định luồng tiếp theo**: Từ node hiện tại, backend đọc danh sách `edges` hoặc `options`. Nếu

node hiện tại là câu hỏi có các phương án trả lời (quick replies), backend sẽ so khớp `text` người dùng gửi với `label` của một edge. Còn nếu chat bot chờ input tự do, có thể áp dụng bộ lọc hoặc NLP đơn giản (ví dụ so khớp từ khóa) để chọn nhánh. Khi tìm được nhánh phù hợp, cập nhật `current_node_id` của user sang node tiếp theo tương ứng. Nếu không tìm thấy nhánh hợp lệ, backend có thể gửi lại yêu cầu người dùng chọn lại hoặc chuyển sang luồng xử lý lỗi.
3. **Cập nhật trạng thái và trả lời**: Sau khi xác định node tiếp theo, backend cập nhật bảng `conversations` với `current_node_id` mới và trả về nội dung của node đó (text và các lựa chọn nếu có) cho n8n. N8n sẽ tiếp tục gửi tin nhắn này tới người dùng. Nếu node mới là câu trả lời cuối hoặc node "xử lý" (ngoài gửi tin nhắn), có thể đánh dấu kết thúc hội thoại (xóa session hoặc để timeout), hoặc quay lại node gốc nếu hội thoại mới bắt đầu.

Để **theo dõi trạng thái của từng user**, ta có thể tạo bảng `conversations` như:

```
CREATE TABLE conversations (
  user_id UUID REFERENCES users(id),
  flow_id INT REFERENCES workflows(id),
  current_node_id TEXT,
  data JSONB,           -- lưu thông tin bổ sung nếu cần (ví dụ form data,
lịch sử)
  updated_at TIMESTAMP DEFAULT NOW(),
  PRIMARY KEY(user_id, flow_id)
);
```

Ví dụ khi user trả lời, backend sẽ thực hiện một truy vấn cập nhật:

```
UPDATE conversations
SET current_node_id = '2', updated_at = NOW()
WHERE user_id = '...' AND flow_id = ...;
```

Thiết kế logic tách rời dữ liệu và xử lý: Workflow (cây JSON) được tách rời với mã xử lý. Backend chỉ cần đọc JSON và tuân theo nhánh được định nghĩa; business logic cụ thể (ví dụ gửi dữ liệu API, validate input) được viết trong code controller/service. Mô hình này tương tự như công cụ Bot Compiler: nó cho phép định nghĩa cây đối thoại trong JSON và có cơ chế quản lý ngữ cảnh (stack) tự động [5]. Nhờ đó, khi bổ sung hoặc sửa luồng hội thoại, không cần thay đổi code – chỉ cần chỉnh JSON. Việc quản lý "context" hay chuyển đổi ngữ cảnh phức tạp (như quay lại thăm một node cũ) có thể được thực hiện bằng ngăn xếp trạng thái (stack) như Bot Compiler đề xuất [5].

Ví dụ một luồng đơn giản: Node 1 hỏi và có 2 tùy chọn, nếu người dùng chọn "Tư vấn", backend chuyển sang Node 2 với câu hỏi tiếp; nếu chọn "Mua hàng", backend chuyển sang Node 3. Trạng thái của user được lưu lại để lần sau tin nhắn mới tiếp tục từ Node 2 hoặc 3.

[5]

API endpoint dành cho n8n và trả lời tin nhắn tiếp theo

Backend cần cung cấp một hoặc một số endpoint REST để n8n gọi khi có tin nhắn từ Facebook Messenger. Ví dụ, ta có thể định nghĩa:

```
POST /api/message
Content-Type: application/json

Request body:
{
  "userId": "facebook_user_id_được_n8n_gửi",
  "text": "Nội dung tin nhắn người dùng gửi",
  "flowId": 123  // (nếu cần thiết, hoặc xác định theo userId)
}
```

Hoặc sử dụng webhook (Webhook node của n8n) nhưng ý tưởng là HTTP Request đến backend. Backend sẽ xử lý như trên: đọc `userId` để tìm hoặc tạo session, duyệt workflow, xác định node tiếp theo.

Định dạng response: Backend trả về JSON chứa thông tin tin nhắn trả lại. Ví dụ:

```
{
  "replyText": "Đây là câu trả lời của chatbot",
  "quickReplies": ["Lựa chọn 1", "Lựa chọn 2"]
}
```

Trong đó `replyText` là chuỗi sẽ gửi cho user, và `quickReplies` là mảng các tùy chọn (nếu có). N8n sau đó sẽ sử dụng node Messenger để gửi tin nhắn với text và nút quick reply tương ứng. Nếu không cần quick replies, mảng đó có thể bỏ trống. Cấu trúc này đảm bảo n8n nhận được đủ thông tin để gọi Facebook Send API.

Xác thực và lỗi: Endpoint nên có cơ chế bảo mật (ví dụ token xác thực hoặc secret trong header) để đảm bảo chỉ n8n (hoặc chính Facebook) mới gọi được. Ngoài ra, backend cần xử lý lỗi (trường hợp JSON sai định dạng, workflow không tồn tại, v.v.) và trả mã lỗi HTTP thích hợp (4xx/5xx). Khi backend trả về tin nhắn cuối cùng của luồng, có thể đặt lại session hoặc để timeout để bắt đầu luồng mới trong lần sau.

Ví dụ sử dụng ExpressJS:

```
app.post('/api/message', async (req, res) => {
  const { userId, text } = req.body;
  // Xác thực (nếu dùng middleware)
  // Xử lý logic ở bước 1-3 phía trên...
  const response = await processMessage(userId, text);
  res.json({
    replyText: response.text,
    quickReplies: response.options
  });
});
```

Như vậy, n8n chỉ cần gọi endpoint này mỗi khi nhận được tin nhắn người dùng và dựa vào JSON response để gửi câu trả lời.

Phát triển sạch và mở rộng (Clean code, Scalable)

Để hệ thống dễ bảo trì và mở rộng, nên áp dụng nguyên tắc **Clean Architecture** và SOLID. Cụ thể, tách rời *business logic* (xử lý luồng hội thoại) khỏi *framework/ngoại vi* (web framework, DB). Ví dụ, dùng NestJS tổ chức code thành các lớp (controllers, services, modules) theo hướng **tách lớp Entities, Use Cases, Adapters, Framework** [3]. Điều này giúp dễ dàng thay đổi phần database, web framework hay thay thành API GraphQL mà không động đến logic chính. Ở mỗi lớp, nên viết test đơn vị (unit test) và tích hợp (integration test).

Các tiêu chí khác:
- **TypeScript**: Sử dụng TypeScript trong cả frontend (React) và backend (Node.js) để tăng độ an toàn kiểu, dễ bảo trì.
- **Thiết kế module**: Ví dụ, một module chuyên về quản lý workflow, một module chuyên về xử lý tin nhắn, một module cho kết nối DB, v.v.
- **ORM/ODM**: Dùng thư viện như **TypeORM** hoặc **Prisma** để làm việc với PostgreSQL, tận dụng khả năng mapping và migration của chúng. Prisma đặc biệt hỗ trợ JSON rất tốt.
- **Cấu hình linh hoạt**: Thông số (DB URL, token, API key) đặt trong biến môi trường, không hard-code.

Về khả năng **mở rộng (scalability)**:
- Node.js vốn có khả năng xử lý nhiều kết nối nhờ cơ chế I/O không chặn, nhưng vẫn cần thiết kế để scale thêm. Các chiến lược mở rộng bao gồm **scale ngang** (chạy nhiều instance backend sau load balancer) hoặc **scale theo dịch vụ** (decompose thành microservices) [6]. Ví dụ, có thể tách riêng service quản lý workflows và service xử lý tin nhắn.
- Sử dụng cơ chế **cluster** của Node.js (hoặc PM2) để tận dụng nhiều CPU cores.
- Dùng **Redis** hoặc Memcached để cache trạng thái session (nếu tải lớn) hoặc làm hàng đợi (queue) để xử lý tác vụ nặng không đồng bộ.
- Triển khai dịch vụ bằng Docker/Kubernetes để dễ mở rộng tự động theo nhu cầu (Horizontal Pod Autoscaling).
- **Auto scaling n8n**: Nếu lượng tin nhắn lớn, có thể chạy n8n ở chế độ queue mode với nhiều worker theo tài liệu của n8n.

Cuối cùng, tuân thủ các **quy ước lập trình** (linter, prettier) và quy trình CI/CD sẽ giúp đội phát triển đóng góp code sạch và an toàn hơn.

[3] [6]

---

[1] Drag and Drop - React Flow
https://reactflow.dev/examples/interaction/drag-and-drop

[2] ReactFlowJsonObject - React Flow
https://reactflow.dev/api-reference/types/react-flow-json-object

[3] Create REST-API with Clean Architecture in NestJS | by Arsy Opraza | Level Up Coding
https://levelup.gitconnected.com/create-rest-api-with-clean-architecture-in-nestjs-d01b58d3f3bc?gi=3aaf724197dd

[4] PostgreSQL: Documentation: 17: 8.14. JSON Types
https://www.postgresql.org/docs/current/datatype-json.html

5 Bot Compiler — Write Dialog tree in JSON and add code only for the functionality. | by Abhishek Vijay | Chatbots Life
https://blog.chatbotslife.com/bot-compiler-write-dialog-tree-in-json-and-add-code-only-for-the-business-logic-7990c739098a?gi=f37ef537b7eb

6 Maximizing Node.js Scalability: Best Practices, Tools, and Patterns
https://www.netguru.com/blog/node-js-scalability